05 Amazon Fine Food Reviews Analysis_Logistic Regression

June 8, 2019

1 Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective: Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative? [Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

2 [1]. Reading Data

2.1 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

In [2]:
```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data point
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5
```

```
# for tsne assignment you can take 5k data points

#filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 50

# 100000 data points for Logistic Regression.
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 ORDER BY T




# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negativ
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (100000, 10)
```

Out[2]:

```
     Id   ProductId          UserId    ProfileName  HelpfulnessNumerator  \
0    10  B00171APVA  A21BT40VZCCYT4  Carol A. Reed                     0
1  1089  B004FD13RW    A1BPLP0BKERV           Paul                     0
2  5703  B009WSNWC4   AMP7K1O84DH1T           ESTY                     0

   HelpfulnessDenominator  Score        Time          Summary  \
0                       0      1  1351209600  Healthy Dog Food
1                       0      1  1351209600    It is awesome.
2                       0      1  1351209600         DELICIOUS

                                               Text
0  This is a very healthy dog food. Good for thei...
1  My partner is very happy with the tea, and is ...
2  Purchased this product at a local store in NY ...
```

In [3]:

```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [4]:

```
print(display.shape)
display.head()
```

```
(80668, 7)
```

```
Out[4]:                UserId   ProductId             ProfileName        Time  Score  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1
        3  #oc-R11O5J5ZVQE25C  B005HG9ET0           Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBE1U    Christopher P. Presta  1348617600      1

                                                        Text  COUNT(*)
        0  Overall its just OK when considering the price...         2
        1  My wife has recurring extreme muscle spasms, u...         3
        2  This coffee is horrible and unfortunately not ...         2
        3  This will be the bottle that you grab from the...         3
        4  I didnt like this coffee. Instead of telling y...         2
```

```
In [5]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[5]:               UserId   ProductId                       ProfileName        Time  \
        80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

               Score                                               Text  COUNT(*)
        80638      5  I was recommended to try green tea extract to ...         5
```

```
In [6]: display['COUNT(*)'].sum()
```

```
Out[6]: 393063
```

3 [2] Exploratory Data Analysis

3.1 [2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

```
In [7]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND UserId="AR5J8UI46CURR"
        ORDER BY ProductID
        """, con)
        display.head()
```

```
Out[7]:        Id   ProductId         UserId    ProfileName  HelpfulnessNumerator  \
        0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
        1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
        2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
```

```
3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2

   HelpfulnessDenominator  Score        Time  \
0                       2      5  1199577600
1                       2      5  1199577600
2                       2      5  1199577600
3                       2      5  1199577600
4                       2      5  1199577600

                             Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  LOACKER QUADRATINI VANILLA WAFERS
3  LOACKER QUADRATINI VANILLA WAFERS
4  LOACKER QUADRATINI VANILLA WAFERS

                                               Text
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

In [9]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
final.shape
```

Out[9]: (71551, 10)

In [10]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]: 71.551

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:
```
      Id   ProductId          UserId              ProfileName  \
0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     3                       1      5  1224892800
1                     3                       2      4  1212883200

                                    Summary  \
0              Bought This for My Son at College
1  Pure cocoa taste with crunchy almonds inside

                                               Text
0  My son loves spaghetti so I didn't hesitate or...
1  It was almost a 'love at first bite' - the per...
```

In [12]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:
```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(71551, 10)
```

Out[13]:
```
1    59256
0    12295
Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

```
In [14]: # printing some random reviews
         sent_0 = final['Text'].values[0]
         print(sent_0)
         print("="*50)

         sent_1000 = final['Text'].values[1000]
         print(sent_1000)
         print("="*50)

         sent_1500 = final['Text'].values[1500]
         print(sent_1500)
         print("="*50)

         sent_4900 = final['Text'].values[4900]
         print(sent_4900)
         print("="*50)
```

```
A charming, rhyming book that describes the circumstances under which you eat (or don't) chick
==================================================
I have one cup a day and it really decreases my night sweats. I'm in amazement at how much it 
==================================================
Strong without being bitter, this is my favorite tea by far.  I was pleasantly surprised recen
==================================================
The Kay's Naturals protein crispy Parmesan  pack of 12 protein chips tasted aweful. I would no
==================================================
```

```
In [15]: # remove urls from text python: https://stackoverflow.com/a/40823105/4084039
         sent_0 = re.sub(r"http\S+", "", sent_0)
         sent_1000 = re.sub(r"http\S+", "", sent_1000)
```

```
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
A charming, rhyming book that describes the circumstances under which you eat (or don't) chick
```

```
In [16]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
         from bs4 import BeautifulSoup

         soup = BeautifulSoup(sent_0, 'lxml')
         text = soup.get_text()
         print(text)
         print("="*50)

         soup = BeautifulSoup(sent_1000, 'lxml')
         text = soup.get_text()
         print(text)
         print("="*50)

         soup = BeautifulSoup(sent_1500, 'lxml')
         text = soup.get_text()
         print(text)
         print("="*50)

         soup = BeautifulSoup(sent_4900, 'lxml')
         text = soup.get_text()
         print(text)
```

```
A charming, rhyming book that describes the circumstances under which you eat (or don't) chick
==================================================
I have one cup a day and it really decreases my night sweats. I'm in amazement at how much it I
==================================================
Strong without being bitter, this is my favorite tea by far.  I was pleasantly surprised recen
==================================================
The Kay's Naturals protein crispy Parmesan  pack of 12 protein chips tasted aweful. I would no
```

```
In [17]: # https://stackoverflow.com/a/47091490/4084039
         import re

         def decontracted(phrase):
             # specific
             phrase = re.sub(r"won't", "will not", phrase)
             phrase = re.sub(r"can\'t", "can not", phrase)

             # general
             phrase = re.sub(r"n\'t", " not", phrase)
```

```
        phrase = re.sub(r"\'re", " are", phrase)
        phrase = re.sub(r"\'s", " is", phrase)
        phrase = re.sub(r"\'d", " would", phrase)
        phrase = re.sub(r"\'ll", " will", phrase)
        phrase = re.sub(r"\'t", " not", phrase)
        phrase = re.sub(r"\'ve", " have", phrase)
        phrase = re.sub(r"\'m", " am", phrase)
        return phrase
```

In [18]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Strong without being bitter, this is my favorite tea by far.  I was pleasantly surprised recen
==================================================
```

In [19]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
A charming, rhyming book that describes the circumstances under which you eat (or don't) chick
```

In [20]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Strong without being bitter this is my favorite tea by far I was pleasantly surprised recently
```

In [21]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'h
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
```

```
                    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mig
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
                    'won', "won't", 'wouldn', "wouldn't"])
```

```
In [22]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_reviews = []
         # tqdm is for printing the status bar
         for sentance in tqdm(final['Text'].values):
             sentance = re.sub(r"http\S+", "", sentance)
             sentance = BeautifulSoup(sentance, 'lxml').get_text()
             sentance = decontracted(sentance)
             sentance = re.sub("\S*\d\S*", "", sentance).strip()
             sentance = re.sub('[^A-Za-z]+', ' ', sentance)
             # https://gist.github.com/sebleier/554280
             sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwo
             preprocessed_reviews.append(sentance.strip())
```

```
100%|| 71551/71551 [00:27<00:00, 2567.60it/s]
```

```
In [23]: preprocessed_reviews[1500]
```

```
Out[23]: 'strong without bitter favorite tea far pleasantly surprised recently showed productio
```

[3.2] Preprocessing Review Summary

```
In [24]: ## Similartly you can do preprocessing for review summary also.
         preprocessed_summaries = []
         # tqdm is for printing the status bar
         for sentence in tqdm(final['Summary'].values):
             #remove URLs
             sentence = re.sub(r"http\S+", "", sentence)
             #remove hml tags
             sentence = BeautifulSoup(sentence, 'lxml').get_text()
             # decontract : won't -> will not
             sentence = decontracted(sentence)
             # remove words with numbers : eg abc123  or just 1234 are both filtered out
             sentence = re.sub("\S*\d\S*", "", sentence).strip()
             # remove special characters
             # if we do not do the step above then this one will convert an 'abc123' to an 'ab
             # the above step will ensure that abc123 is completely removed from our result se
             sentence = re.sub('[^A-Za-z]+', ' ', sentence)
             # https://gist.github.com/sebleier/554280
             # also performing stemming here using Snowball stemmer.
             #if we stem then  pre-trained Google W2V may fail to find a vector for tasti (the
             #sentence = ' '.join(sno.stem(e.lower()) for e in sentence.split() if e.lower() no
             sentence = ' '.join(e.lower() for e in sentence.split() if e.lower() not in stopwo
             preprocessed_summaries.append(sentence.strip())
```

```
100%|| 71551/71551 [00:17<00:00, 4007.48it/s]
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [0]: #BoW
        count_vect = CountVectorizer() #in scikit-learn
        count_vect.fit(preprocessed_reviews)
        print("some feature names ", count_vect.get_feature_names()[:10])
        print('='*50)

        final_counts = count_vect.transform(preprocessed_reviews)
        print("the type of count vectorizer ",type(final_counts))
        print("the shape of out text BOW vectorizer ",final_counts.get_shape())
        print("the number of unique words ", final_counts.get_shape()[1])
```

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', 'abby', 'abdomina
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

5.2 [4.2] Bi-Grams and n-Grams.

```
In [0]: #bi-gram, tri-gram and n-gram

        #removing stop words like "not" should be avoided before building n-grams
        # count_vect = CountVectorizer(ngram_range=(1,2))
        # please do read the CountVectorizer documentation http://scikit-learn.org/stable/modu

        # you can choose these numebrs min_df=10, max_features=5000, of your choice
        count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
        final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
        print("the type of count vectorizer ",type(final_bigram_counts))
        print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
        print("the number of unique words including both unigrams and bigrams ", final_bigram_
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

5.3 [4.3] TF-IDF

```
In [0]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
        tf_idf_vect.fit(preprocessed_reviews)
```

```
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.g
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get',
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

5.4 [4.4] Word2Vec

In [0]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [0]:
```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
```

```
        print(w2v_model.wv.most_similar('worst'))

    elif want_to_use_google_w2v and is_your_ram_gt_16g:
        if os.path.isfile('GoogleNews-vectors-negative300.bin'):
            w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bi
            print(w2v_model.wv.most_similar('great'))
            print(w2v_model.wv.most_similar('worst'))
        else:
            print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful', 0.99460321664810
==========================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn', 0.99927508831O
```

```
In [0]: w2v_words = list(w2v_model.wv.vocab)
        print("number of words that occured minimum 5 times ",len(w2v_words))
        print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby
```

5.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

```
In [0]: # average Word2Vec
        # compute average word2vec for each review.
        sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
        for sent in tqdm(list_of_sentance): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to
            cnt_words =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words:
                    vec = w2v_model.wv[word]
                    sent_vec += vec
                    cnt_words += 1
            if cnt_words != 0:
                sent_vec /= cnt_words
            sent_vectors.append(sent_vec)
        print(len(sent_vectors))
        print(len(sent_vectors[0]))
```

```
100%|| 4986/4986 [00:03<00:00, 1330.47it/s]


4986
50
```

[4.4.1.2] TFIDF weighted W2v

```
In [0]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        model = TfidfVectorizer()
        tf_idf_matrix = model.fit_transform(preprocessed_reviews)
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [0]: # TF-IDF weighted Word2Vec
        tfidf_feat = model.get_feature_names() # tfidf words/col-names
        # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

        tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this li
        row=0;
        for sent in tqdm(list_of_sentance): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length
            weight_sum =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words and word in tfidf_feat:
                    vec = w2v_model.wv[word]
        #             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                    # to reduce the computation we are
                    # dictionary[word] = idf value of word in whole courpus
                    # sent.count(word) = tf valeus of word in this review
                    tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                    sent_vec += (vec * tf_idf)
                    weight_sum += tf_idf
            if weight_sum != 0:
                sent_vec /= weight_sum
            tfidf_sent_vectors.append(sent_vec)
            row += 1
```

```
100%|| 4986/4986 [00:20<00:00, 245.63it/s]
```

6 [5] Assignment 5: Apply Logistic Regression

```
<li><strong>Apply Logistic Regression on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vector
    </ul>
</li>
<br>
<li><strong>Hyper paramter tuning (find best hyper parameters corresponding the algorithm that
    <ul>
```

```
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </ul>
</li>
<br>
<li><strong>Pertubation Test</strong>
    <ul>
<li>Get the weights W after fit your model with the data X i.e Train data.</li>
<li>Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse
```

matrix, X.data+=e)

```
<li>Fit the model again on data X' and get the weights W'</li>
<li>Add a small eps value(to eliminate the divisible by zero error) to W and W i.e
```

W=W+10^-6 and W′ = W′+10^-6

```
<li>Now find the % change between W and W' (| (W-W') / (W) |)*100)</li>
<li>Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in t
<li> Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is suc
        <li> Print the feature names whose % change is more than a threshold x(in our example i
    </ul>
</li>
<br>
<li><strong>Sparsity</strong>
    <ul>
<li>Calculate sparsity on weight vector obtained after using L1 regularization</li>
    </ul>
</li>
<br><font color='red'>NOTE: Do sparsity and multicollinearity for any one of the vectorizers. B
<br>
<br>
<li><strong>Feature importance</strong>
    <ul>
<li>Get top 10 important features for both positive and negative classes separately.</li>
    </ul>
</li>
<br>
<li><strong>Feature engineering</strong>
    <ul>
<li>To increase the performance of your model, you can also experiment with with feature engine
        <ul>
        <li>Taking length of reviews as another feature.</li>
        <li>Considering some features from review summary as well.</li>
    </ul>
    </ul>
</li>
<br>
```

```
<li><strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

7 Applying Logistic Regression

In [25]:
```python
#concatenate the review summary with review text.
preprocessed_text_n_summary = [(preprocessed_reviews[i] + ' ' +  preprocessed_summarie
print(preprocessed_text_n_summary[0])
print(len(preprocessed_text_n_summary))
```

```
charming rhyming book describes circumstances eat not chicken soup rice month month sounds lik
71551
```

In [26]:
```python
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sk
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from tqdm import tqdm

from sklearn.metrics import roc_auc_score
```

```
import matplotlib.pyplot as plt
import math
import seaborn as sns

C = [10**-4, 10**-2, 10**0, 10**2, 10**4]

Y = final['Score']

# adding the summary text to get additional features
X = preprocessed_text_n_summary

# Train on oldest data (eg. Now - 90 days), CV on somewhat recent  data (eg. Now - 30
# doing a time series split:  swapped the test and train as our data is in DESCENDING
# and we want X_test to have the most recent data.
X_test, X_train, y_test, y_train = train_test_split(X, Y, test_size=0.77, shuffle=Fals

# trying a random shuffle to see if it performs better
#X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Tr
# time series splitting
# not splitting further into CV and Train because we plan to use GridSearch with 3 fo
# we pass in entire X_train to GridSearch.
#X_cv, X_train, y_cv, y_train = train_test_split(X_train, y_train, test_size=0.77, sh

print('X_train size=' , len(X_train))
print('X_test size=', len(X_test))
print('y_train class counts')
print(y_train.value_counts())
print('y_test class counts')
print(y_test.value_counts())
```

```
X_train size= 55095
X_test size= 16456
y_train class counts
1    45830
0     9265
Name: Score, dtype: int64
y_test class counts
1    13426
0     3030
Name: Score, dtype: int64
```

In [32]:
```
import math
import operator

def predictAndPlot(X_train, y_train, X_test, y_test, clf):
    train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(X_train)[
    test_fpr, test_tpr, thresholds = roc_curve(y_test, clf.predict_proba(X_test)[:,1]
```

```
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.xlabel("fpr")
    plt.ylabel("tpr")
    plt.title("ERROR PLOTS")
    plt.show()
    print("="*100)

    cmTrain = confusion_matrix(y_train, clf.predict(X_train))
    #print("Test confusion matrix")
    cmTest= confusion_matrix(y_test, clf.predict(X_test))



    plt.figure(figsize=(10,5))
    trainx= plt.subplot(1, 3, 1)
    sns.heatmap(cmTrain, annot=True, ax = trainx, cmap='Blues', fmt='g'); #annot=True
    # labels, title and ticks
    trainx.set_xlabel('Actual');trainx.set_ylabel('Predicted');
    trainx.set_title('Train Confusion Matrix');
    trainx.xaxis.set_ticklabels(['negative', 'positive']); trainx.yaxis.set_ticklabels

    testx= plt.subplot(1, 3, 3)
    sns.heatmap(cmTest, annot=True, ax = testx, cmap='Blues', fmt='g'); #annot=True t
    # labels, title and ticks
    testx.set_xlabel('Actual');testx.set_ylabel('Predicted');
    testx.set_title('Test Confusion Matrix');
    testx.xaxis.set_ticklabels(['negative', 'positive']); testx.yaxis.set_ticklabels(

def sort_tuplelist(tupleList):
    tupleList.sort(key = operator.itemgetter(1))
    return tupleList

def train_and_plot_auc(reg, C, X_train_data, y_train_data):
    parameters = [{'C': C}]
    clf = LogisticRegression(penalty=reg,class_weight='balanced')
    clf = GridSearchCV(clf, parameters, cv=3, scoring='roc_auc')
    clf.fit(X_train_data, y_train_data)

    train_auc= clf.cv_results_['mean_train_score']
    train_auc_std= clf.cv_results_['std_train_score']
    cv_auc = clf.cv_results_['mean_test_score']
    cv_auc_std= clf.cv_results_['std_test_score']
    plt.plot(C, train_auc, label='Train AUC')
    # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
    plt.gca().fill_between(C, train_auc - train_auc_std,train_auc + train_auc_std,alph
```

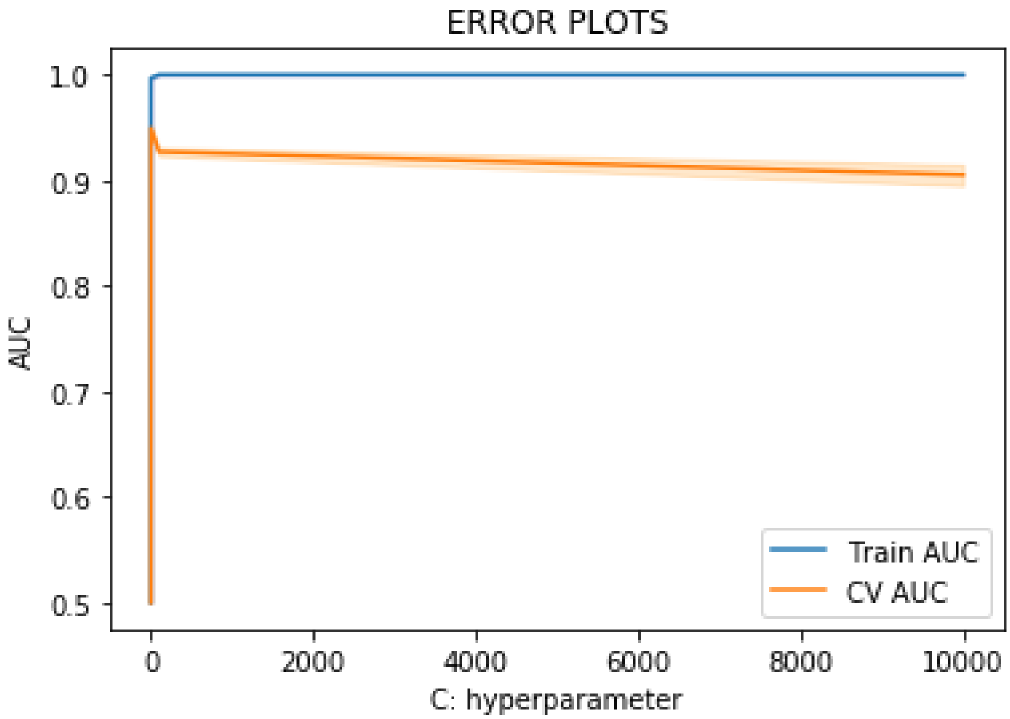
```
    plt.plot(C, cv_auc, label='CV AUC')
    # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
    plt.gca().fill_between(C, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color
    plt.legend()
    plt.xlabel("C: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
    return clf
```

7.1 [5.1] Logistic Regression on BOW, SET 1

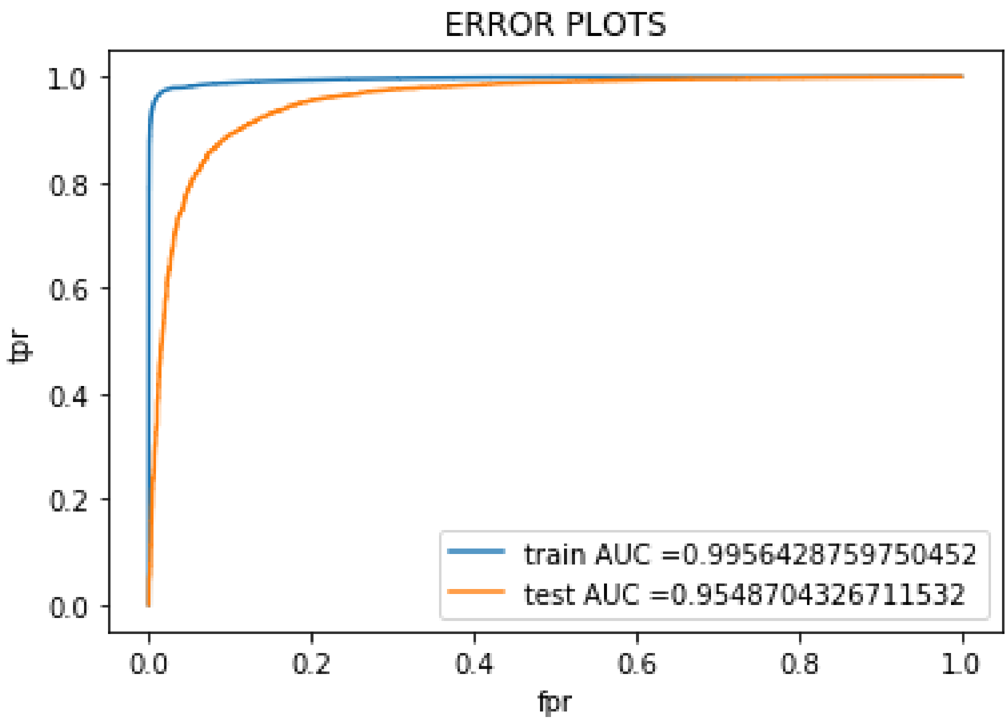
7.1.1 [5.1.1] Applying Logistic Regression with L1 regularization on BOW, SET 1

In [28]:
```
# Please write all the code with proper documentation
from sklearn.feature_extraction.text import CountVectorizer
import numpy
from sklearn.metrics import confusion_matrix


vectorizer = CountVectorizer()

# While vectorizing your data, apply the method fit_transform() on you train data,
# and apply the method transform() on cv/test data.
# THE VOCABULARY SHOULD BUILT ONLY WITH THE WORDS OF TRAIN DATA
vectorizer.fit(X_train)

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_test_bow = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_test_bow.shape, y_test.shape)
print(type(X_train_bow))


clf_l1 = train_and_plot_auc('l1',C,X_train_bow, y_train)
y_pred = clf_l1.predict(X_test_bow)
print('Confusion Matrix : \n' + str(confusion_matrix(y_test,y_pred)))

predictAndPlot(X_train_bow, y_train, X_test_bow, y_test, clf_l1)
```

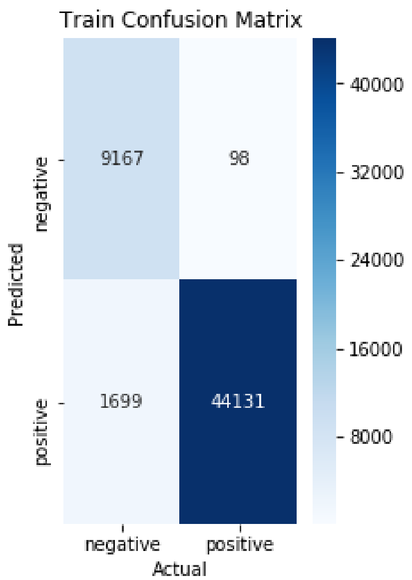
```
After vectorizations
(55095, 45487) (55095,)
(16456, 45487) (16456,)
<class 'scipy.sparse.csr.csr_matrix'>
```

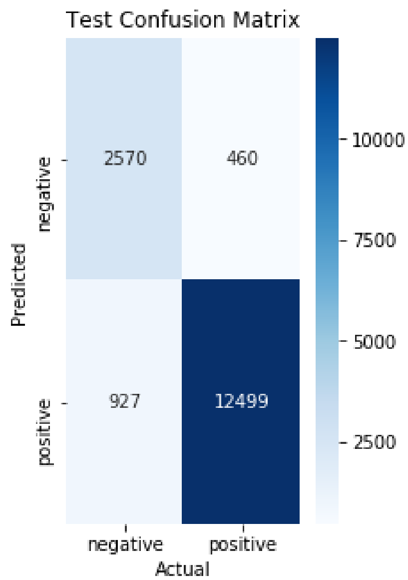



```
Confusion Matrix :
[[ 2570   460]
 [  927 12499]]
```


ERROR PLOTS
1.0
0.8
0.6
0.4
0.2
0.0
tpr
0.0
0.2
0.4
0.6
0.8
1.0
fpr
train AUC =0.9956428759750452
test AUC =0.9548704326711532

==


Train Confusion Matrix
9167
98
1699
44131
negative
positive
Predicted
negative
positive
Actual
40000
32000
24000
16000
8000


Test Confusion Matrix
2570
460
927
12499
negative
positive
Predicted
negative
positive
Actual
10000
7500
5000
2500

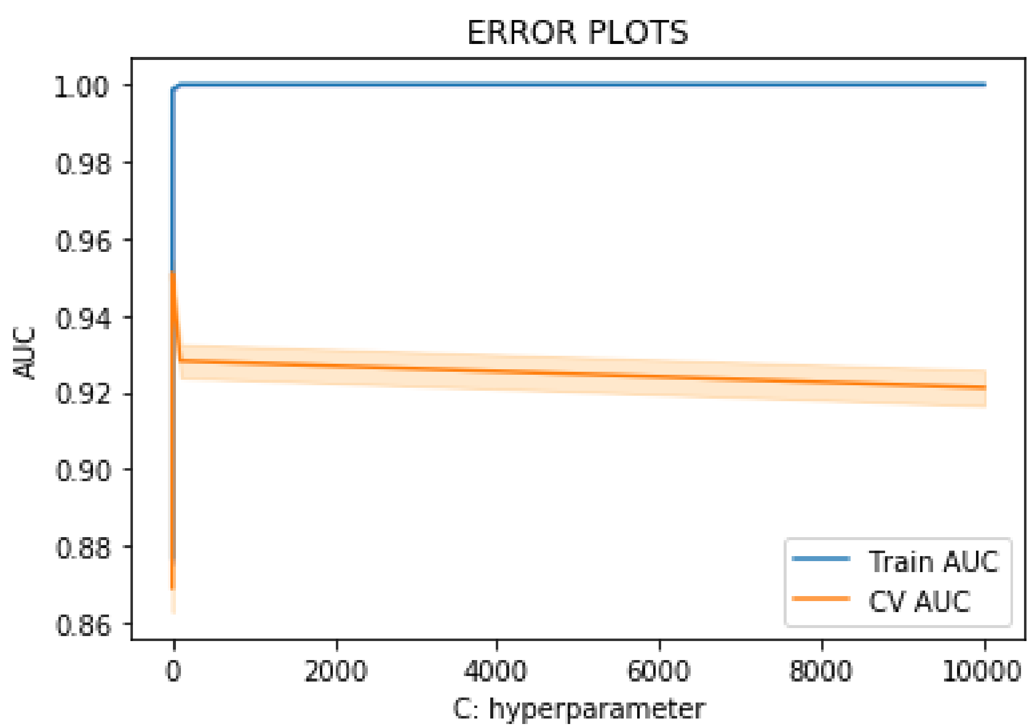
[5.1.1.1] Calculating sparsity on weight vector obtained using L1 regularization on BOW, SET 1

```
In [29]: # Please write all the code with proper documentation
         print(clf_l1.score(X_test_bow, y_test))
         bestC = clf_l1.best_params_
         print(bestC)
         w = clf_l1.best_estimator_.coef_
         print(np.count_nonzero(w))
```

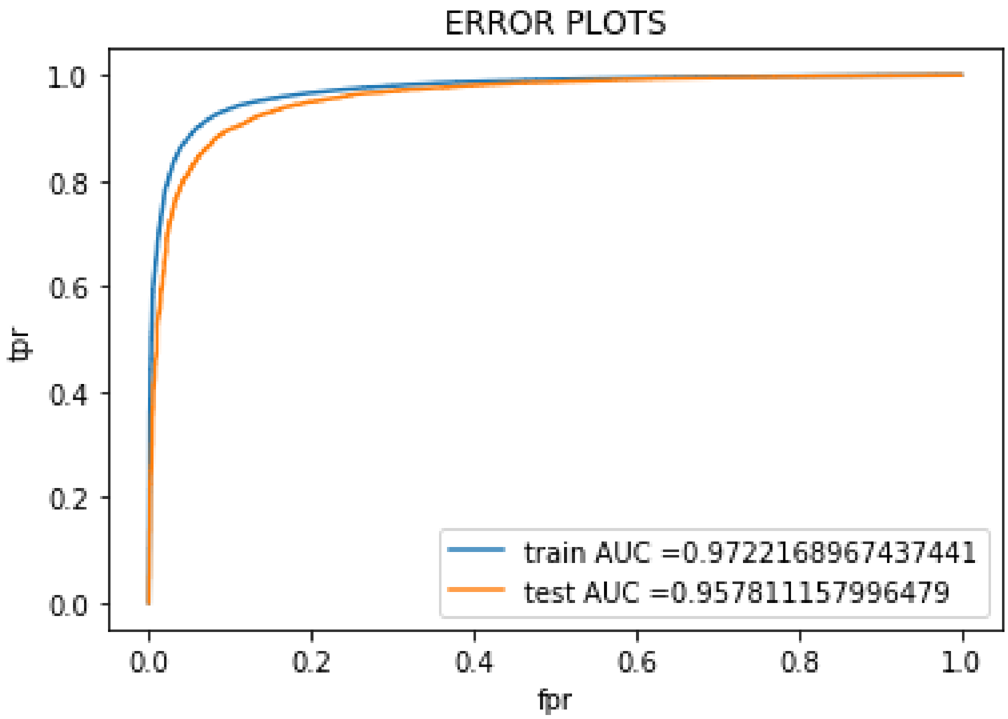
```
0.9548704326711532
{'C': 1}
4585
```

7.1.2 [5.1.2] Applying Logistic Regression with L2 regularization on BOW, SET 1

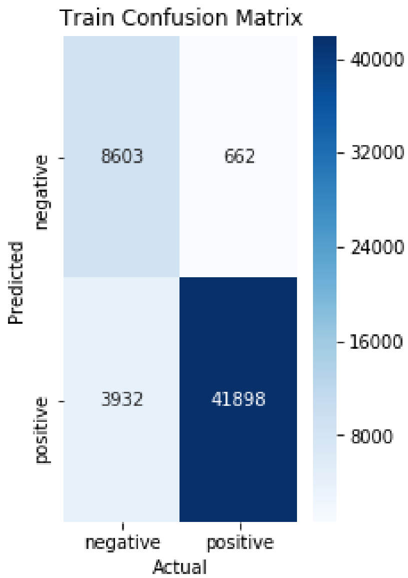
```
In [30]: # Please write all the code with proper documentation
         clf = train_and_plot_auc('l2',C,X_train_bow, y_train)
         y_pred = clf.predict(X_test_bow)
         print('Confusion Matrix : \n' + str(confusion_matrix(y_test,y_pred)))

         predictAndPlot(X_train_bow, y_train, X_test_bow, y_test, clf)
         print(clf.score(X_test_bow, y_test))
         bestC = clf.best_params_
         print(bestC)
         w = clf.best_estimator_.coef_
         print(np.count_nonzero(w))
```

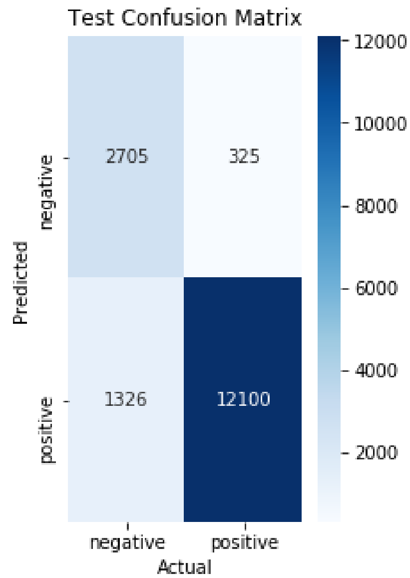


```
Confusion Matrix :
[[ 2705   325]
 [ 1326 12100]]
```




```
================================================================================================
0.957811157996479
{'C': 0.01}
45487
```



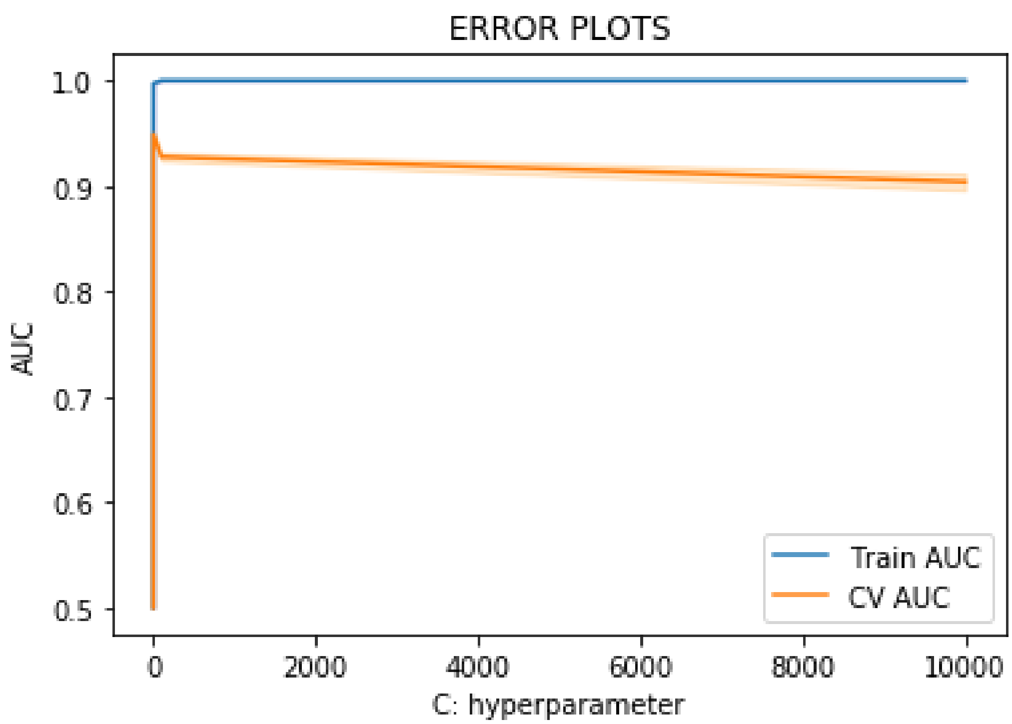


[5.1.2.1] Performing pertubation test (multicollinearity check) on BOW, SET 1

In [48]:
```
import numpy as np
# Please write all the code with proper documentation
#Get the weights W after fit your model with the data X i.e Train data.
#Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matr
#Fit the model again on data X' and get the weights W'
#Add a small eps value(to eliminate the divisible by zero error) to W and W i.e W=W+1
#Now find the % change between W and W' (| (W-W') / (W) |)*100)
#Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden ri
#Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there
#Print the feature names whose % change is more than a threshold x(in our example it'

# let's use L1 reg reults here
w = clf_l1.best_estimator_.coef_
weights = w[0]
print(type(weights))
print(weights)
len(weights)

#feature_names = vectorizer.get_feature_names()
#dictionary = dict(zip(feature_names, weights))
#print(dictionary)

print(type(X_train_bow))
X_train_bow_perturbed = X_train_bow.asfptype()
# add noise to X_train_bow
X_train_bow_perturbed.data += 0.10
```

```python
#train again
clf_l1 = train_and_plot_auc('l1',C,X_train_bow_perturbed, y_train)
#
wp = clf_l1.best_estimator_.coef_
w_perturbed = wp[0]

weights = weights + 10**-6
w_perturbed = w_perturbed + 10**-6

percentage_change = (np.absolute((weights - w_perturbed)/ weights)) * 100
print(percentage_change)
print(np.argmax(percentage_change))
print(np.argmin(percentage_change))
print(len(percentage_change))
```

```
<class 'numpy.ndarray'>
[0. 0. 0. ... 0. 0. 0.]
<class 'scipy.sparse.csr.csr_matrix'>
```




```
[0. 0. 0. ... 0. 0. 0.]
10246
```

```
0
45487
```

```
In [49]: #https://stackoverflow.com/questions/26070514/how-do-i-get-the-index-of-a-specific-pe
         percentile_array = []
         percential_array_index = []
         for a in range(0,110,10):
             percentile_array.append(np.percentile(percentage_change, a))
             percential_array_index.append(abs(percentage_change-np.percentile(percentage_chang

         print(percentile_array)
         print(percential_array_index)
```

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8377813657108503, 134019737.09577058]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 6454, 10246]
```

```
In [50]: #the sudden change in value occurs between 90th and 100th percentile.
         for a in range(90,101,1):
             print(np.percentile(percentage_change, a))
```

```
0.8377813657108503
2.2526145081075324
3.570121334149716
4.904033269142897
6.310717205463914
7.929577817202095
10.404790832644618
15.966493985190866
30.76651415499005
99.04679598731391
134019737.09577058
```

```
In [51]: # the sudden change is actually between 99th percentile and 100th percentile
         z = 0.1
         p = 99.0
         for a in range(0,11):
             print('percentile=', p)
             print('value=', np.percentile(percentage_change, p))
             print('index=', abs(percentage_change-np.percentile(percentage_change,p,interpola
             p += z
```

```
percentile= 99.0
value= 99.04679598731391
index= 31542
percentile= 99.1
```

```
value= 100.00392517700179
index= 25282
percentile= 99.19999999999999
value= 113.83395099829991
index= 5776
percentile= 99.29999999999998
value= 155.67658330113045
index= 17876
percentile= 99.39999999999998
value= 247.11166799681877
index= 24671
percentile= 99.49999999999997
value= 948.4522383198621
index= 34452
percentile= 99.59999999999997
value= 262679.03022406006
index= 4900
percentile= 99.69999999999996
value= 1762870.5483640572
index= 22812
percentile= 99.79999999999995
value= 4617951.6556952745
index= 34966
percentile= 99.89999999999995
value= 9810310.274979506
index= 32532
percentile= 99.99999999999994
value= 134019737.09516817
index= 10246
```

In [53]:
```python
# the sudden change is from value  948.452 to 262679.0302
# feature names with value > 948.452 are
collinear_feature_indexes = [i for i,v in enumerate(percentage_change) if v > 948.452]
collinear_feature_names = [vectorizer.get_feature_names()[i] for i in collinear_featu
print(collinear_feature_names)
print(len(collinear_feature_names))
```

```
['actual', 'advertising', 'agent', 'aggravating', 'al', 'amounts', 'andyou', 'aobut', 'aquire'
228
```

7.1.3 [5.1.3] Feature Importance on BOW, SET 1

In [33]:
```python
#After you compute the weight vector, select the indexes of the highest 10 and lowest
#in the weight vector. The feature names for BOW, TF-IDF can be obtained using get_fe
#Now from those features obtained, pick the features that are corresponding to the in
#and lowest coefficients.
```

```
#Features associated with top 10 highest coefficients are the features that contribut
#Features associated with top 10 lowest coefficients are the features that contribute
#using the results of L2 Regularized classfier.
coefficients = clf.best_estimator_.coef_
w_star = coefficients[0]
feature_names = vectorizer.get_feature_names()
tuples = list(zip(feature_names, w_star))
sorted_tuples = sort_tuplelist(tuples)
#print(sorted_tuples)
```

[5.1.3.1] Top 10 important features of positive class from SET 1

In [41]:
```
positive_features = sorted_tuples[-10:]
print(positive_features)
```

```
[('delicious', 1.004626028885019), ('perfect', 0.9025355326169747), ('great', 0.89362043839982
```

[5.1.3.2] Top 10 important features of negative class from SET 1

In [42]:
```
# Please write all the code with proper documentation
negative_features = sorted_tuples[:10]
print(negative_features)
```

```
[('horrible', -0.6540589934837376), ('awful', -0.6614138501535496), ('terrible', -0.70591149648
```

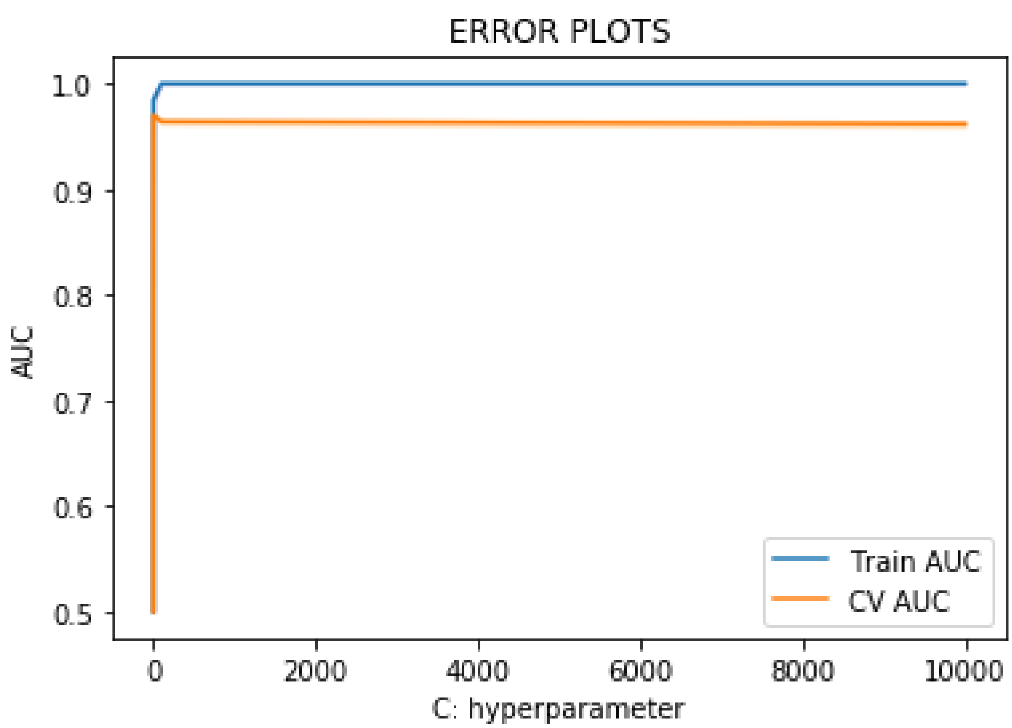
7.2 [5.2] Logistic Regression on TFIDF, SET 2

7.2.1 [5.2.1] Applying Logistic Regression with L1 regularization on TFIDF, SET 2

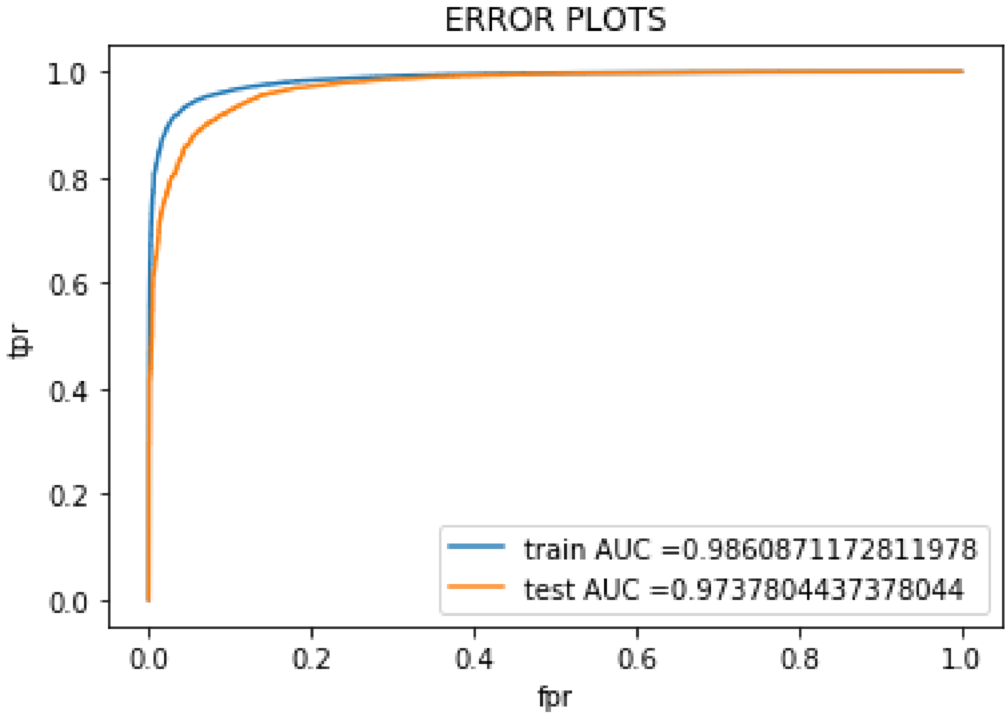
In [34]:
```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from tqdm import tqdm


vectorizer = TfidfVectorizer(ngram_range=(1,2), min_df=10)
vectorizer.fit(X_train)

# we use the fitted vectorizer to convert the text to vector
X_train_tfidf = vectorizer.transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_tfidf.shape, y_train.shape)
print(X_test_tfidf.shape, y_test.shape)
```

```
print(type(X_train_tfidf))



clf = train_and_plot_auc('l1',C,X_train_tfidf, y_train)
y_pred = clf.predict(X_test_tfidf)
print('Confusion Matrix : \n' + str(confusion_matrix(y_test,y_pred)))

predictAndPlot(X_train_tfidf, y_train, X_test_tfidf, y_test, clf)
print(clf.score(X_test_tfidf, y_test))
bestC = clf.best_params_
print(bestC)
w = clf.best_estimator_.coef_
print(np.count_nonzero(w))
```

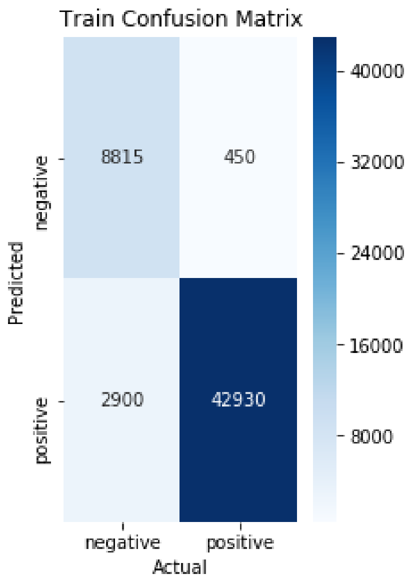
```
After vectorizations
(55095, 34572) (55095,)
(16456, 34572) (16456,)
<class 'scipy.sparse.csr.csr_matrix'>
```

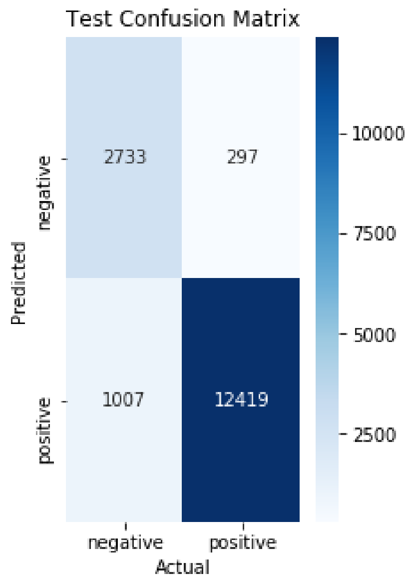



```
Confusion Matrix :
[[ 2733   297]
 [ 1007 12419]]
```



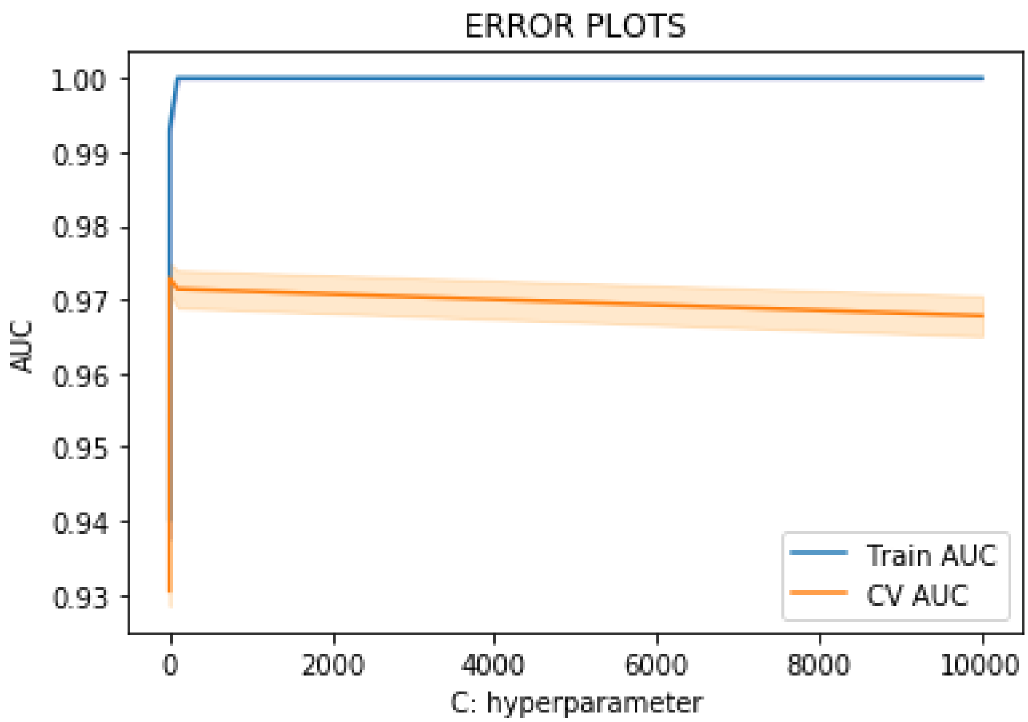

```
==========================================================================================
0.9737804437378044
{'C': 1}
1533
```



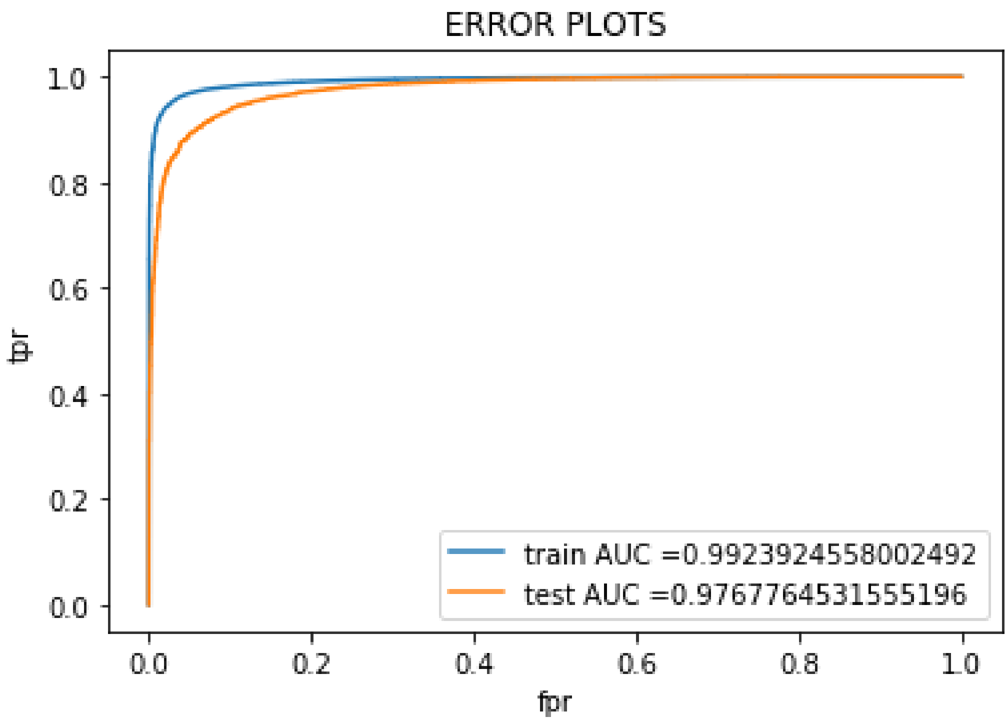


7.2.2 [5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

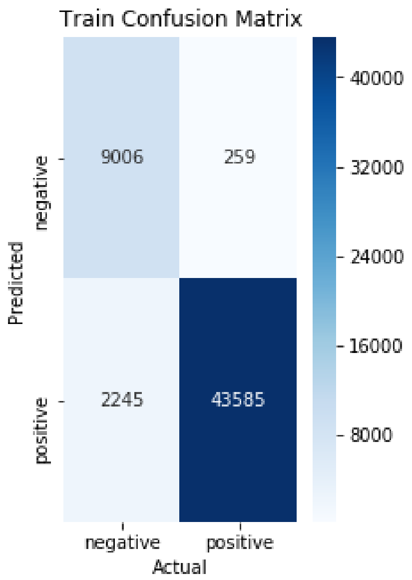
In [35]:
```
# Please write all the code with proper documentation
clf = train_and_plot_auc('l2',C,X_train_tfidf, y_train)
y_pred = clf.predict(X_test_tfidf)
print('Confusion Matrix : \n' + str(confusion_matrix(y_test,y_pred)))

predictAndPlot(X_train_tfidf, y_train, X_test_tfidf, y_test, clf)
print(clf.score(X_test_tfidf, y_test))
bestC = clf.best_params_
print(bestC)
w = clf.best_estimator_.coef_
print(np.count_nonzero(w))
```

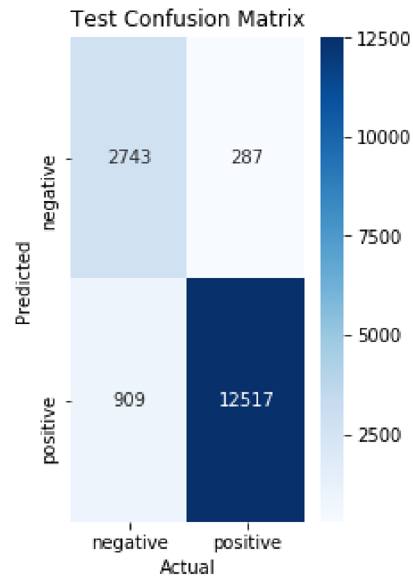



```
Confusion Matrix :
[[ 2743   287]
 [  909 12517]]
```




```
================================================================================
0.9767764531555196
{'C': 1}
34572
```





7.2.3 [5.2.3] Feature Importance on TFIDF, SET 2

```
In [36]: feature_names = vectorizer.get_feature_names()
         coefficients = clf.best_estimator_.coef_
         w_star = coefficients[0]

         tuples = list(zip(feature_names, w_star))
         sorted_tuples = sort_tuplelist(tuples)
```

[5.2.3.1] Top 10 important features of positive class from SET 2

```
In [39]: positive_features = sorted_tuples[-10:]
         print(positive_features)
```

```
[('wonderful', 6.503125165934264), ('favorite', 6.533597294901277), ('loves', 6.69601624487120
```

[5.2.3.2] Top 10 important features of negative class from SET 2

```
In [40]: negative_features = sorted_tuples[:10]
         print(negative_features)
```

```
[('not', -10.76939812252225), ('disappointed', -9.748632037483171), ('not good', -7.730860534
```

7.3 [5.3] Logistic Regression on AVG W2V, SET 3

```
In [41]: from gensim.models import Word2Vec
         from gensim.models import KeyedVectors


         def computeAvgW2V(X):
             list_of_sentence_train=[]
             for sentence in X:
                 list_of_sentence_train.append(sentence.split())

             # this line of code trains your w2v model on the give list of sentances
             w2v_model=Word2Vec(list_of_sentence_train,min_count=20,size=50, workers=4,iter=15
             w2v_words = list(w2v_model.wv.vocab)

             # average Word2Vec
             # compute average word2vec for each review.
             sent_vectors_train = [] # the avg-w2v for each sentence/review is stored in this
             for sent in tqdm(list_of_sentence_train): # for each review/sentence
                 sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might ne
                 cnt_words =0; # num of words with a valid vector in the sentence/review
                 for word in sent: # for each word in a review/sentence
                     if word in w2v_words:
                         vec = w2v_model.wv[word]
```

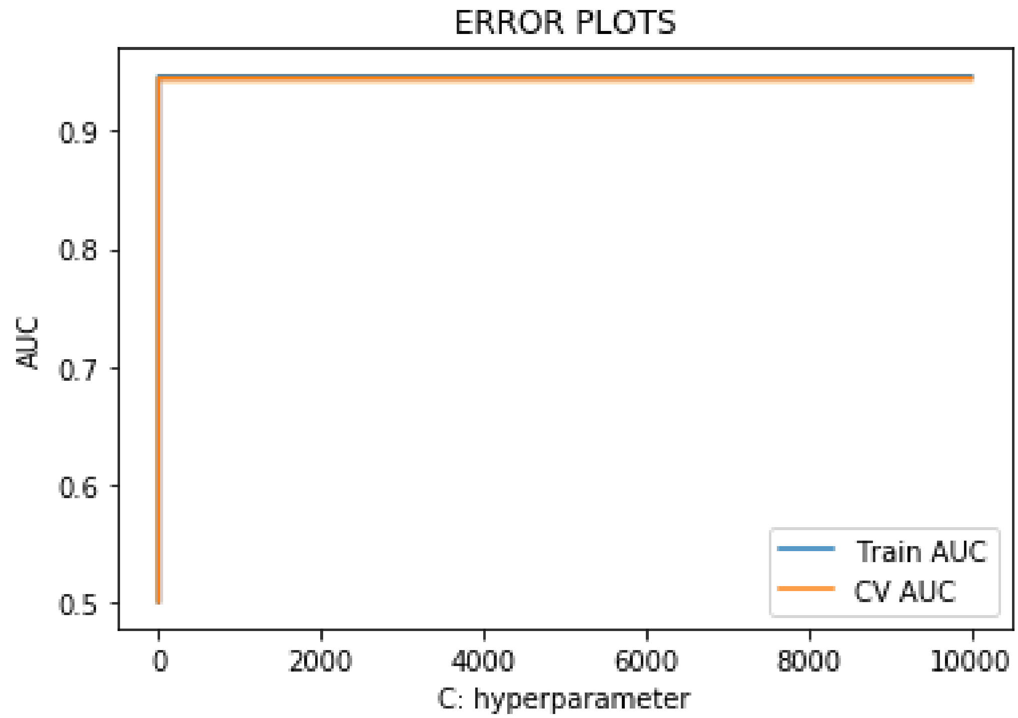
```
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors_train.append(sent_vec)
    sent_vectors_train = np.array(sent_vectors_train)
    print(sent_vectors_train.shape)
    print(sent_vectors_train[0])
    return sent_vectors_train
```

7.3.1 [5.3.1] Applying Logistic Regression with L1 regularization on AVG W2V SET 3

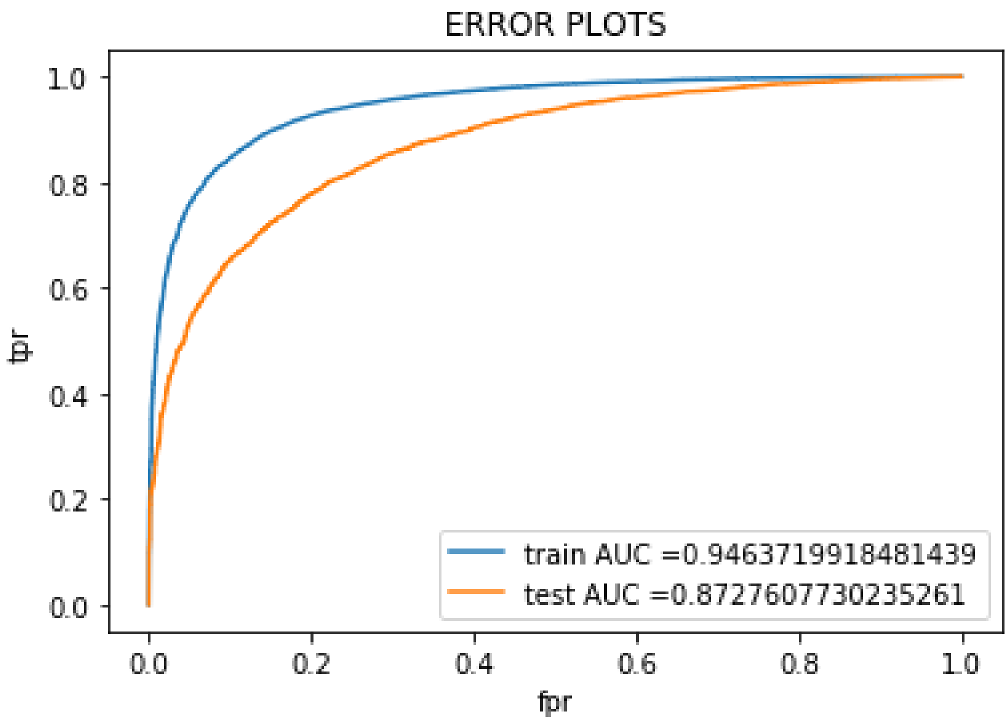
```
In [42]: X_train_w2vAvg = computeAvgW2V(X_train)
         X_test_w2vAvg = computeAvgW2V(X_test)


         clf = train_and_plot_auc('l1',C,X_train_w2vAvg, y_train)
         y_pred = clf.predict(X_test_w2vAvg)
         print('Confusion Matrix : \n' + str(confusion_matrix(y_test,y_pred)))

         predictAndPlot(X_train_w2vAvg, y_train, X_test_w2vAvg, y_test, clf)
         print(clf.score(X_test_w2vAvg, y_test))
         bestC = clf.best_params_
         print(bestC)
         w = clf.best_estimator_.coef_
         print(np.count_nonzero(w))
```

```
100%|| 55095/55095 [01:20<00:00, 681.46it/s]


(55095, 50)
[-0.18921404 -0.041131   -0.31436349 -0.24266228  0.58941854  0.49746976
 -0.64779979 -0.04579963 -0.23828633 -0.30898747  0.27114164 -0.30109271
 -0.48232371  0.05522433 -0.09489444  0.33604911  0.45211268  0.0760485
  0.5149875   0.41913828  0.0344959   0.55790203  0.32099965  0.36027644
  0.41594385  0.24965249 -0.54011213 -0.06092552  0.15357676  0.02713726
  0.3246661   0.44625689 -0.62664956  0.85482717 -0.25234905  0.42406213
  0.88341015 -0.38683618  0.42223536  0.06676471 -0.26478488  0.41117351
 -0.568865   -0.56247315 -0.1730448   0.13604499 -0.18998017 -0.05540112
  0.59512974  0.20191685]


100%|| 16456/16456 [00:17<00:00, 935.47it/s]


(16456, 50)
[ 0.60988894 -0.7913202   0.23412916 -0.11136134 -0.17425462 -0.11613596
  0.49990036 -0.08805562  0.31801226  0.09938592 -0.36080722 -0.05560918
  0.14462608 -0.08105637  0.68985326  0.21035072  0.14453829  0.46982845
```

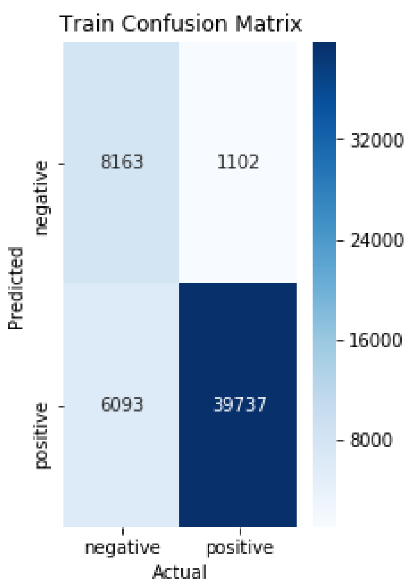
```
  0.1211412  -0.01806533 -0.06238089  0.22374712 -0.14412294 -0.18814267
 -1.14709838  0.12458374 -0.49951372 -0.2119959  -0.6634607   0.38621513
 -0.18790756 -0.30233719  0.10153362 -0.33494587  0.18856375  0.5230891
  0.56783575 -0.21861941  0.56829925 -0.51153512 -0.18802216 -0.25038399
 -0.26520994 -0.79293803  0.14538662 -0.33775963 -0.12152057 -0.35491055
 -0.08031984  0.2104208 ]
```

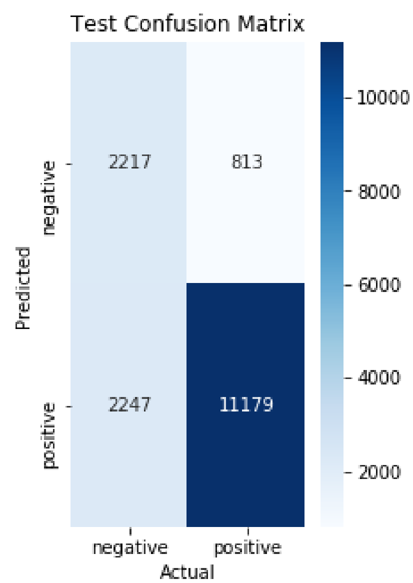



```
Confusion Matrix :
[[ 2217   813]
 [ 2247 11179]]
```



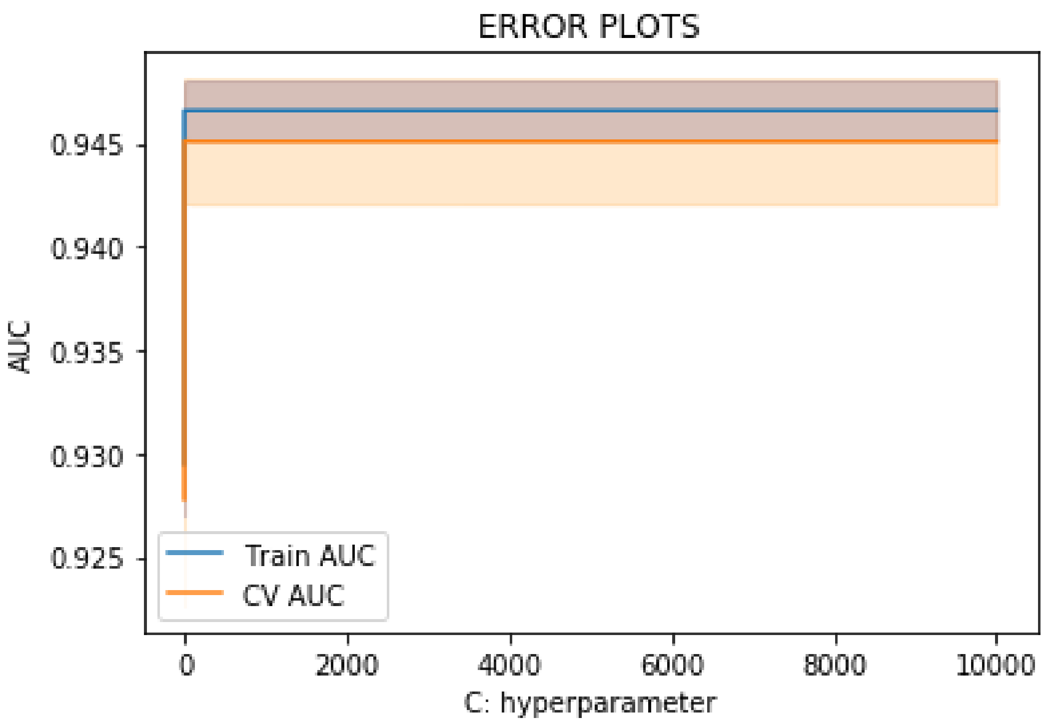

```
==========================================================================================
0.8727607730235261
{'C': 1}
50
```



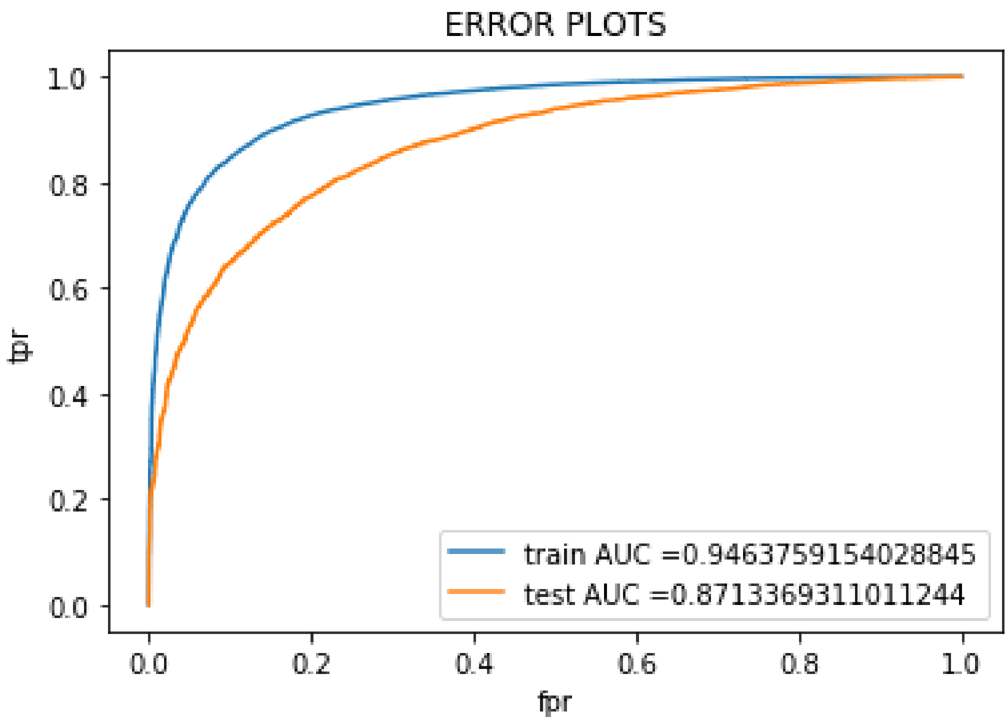


7.3.2 [5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

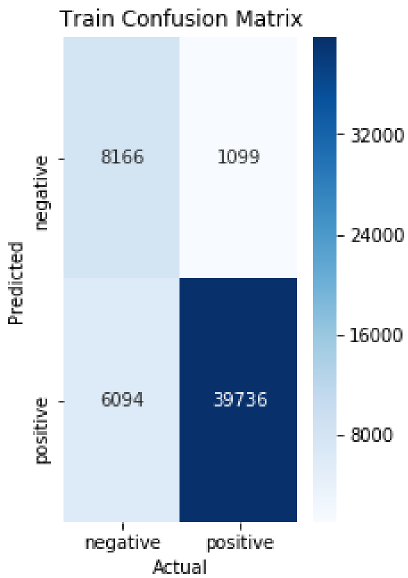
In [43]:
```
clf = train_and_plot_auc('l2',C,X_train_w2vAvg, y_train)
y_pred = clf.predict(X_test_w2vAvg)
print('Confusion Matrix : \n' + str(confusion_matrix(y_test,y_pred)))

predictAndPlot(X_train_w2vAvg, y_train, X_test_w2vAvg, y_test, clf)
print(clf.score(X_test_w2vAvg, y_test))
bestC = clf.best_params_
print(bestC)
w = clf.best_estimator_.coef_
print(np.count_nonzero(w))
```

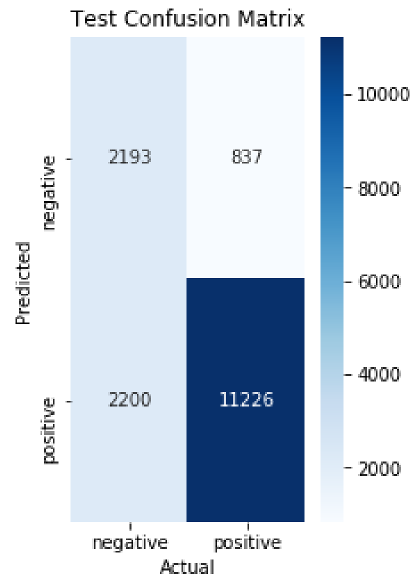



```
Confusion Matrix :
[[ 2193   837]
 [ 2200 11226]]
```




```
========================================================================================
0.8713369311011244
{'C': 1}
50
```





7.4 [5.4] Logistic Regression on TFIDF W2V, SET 4

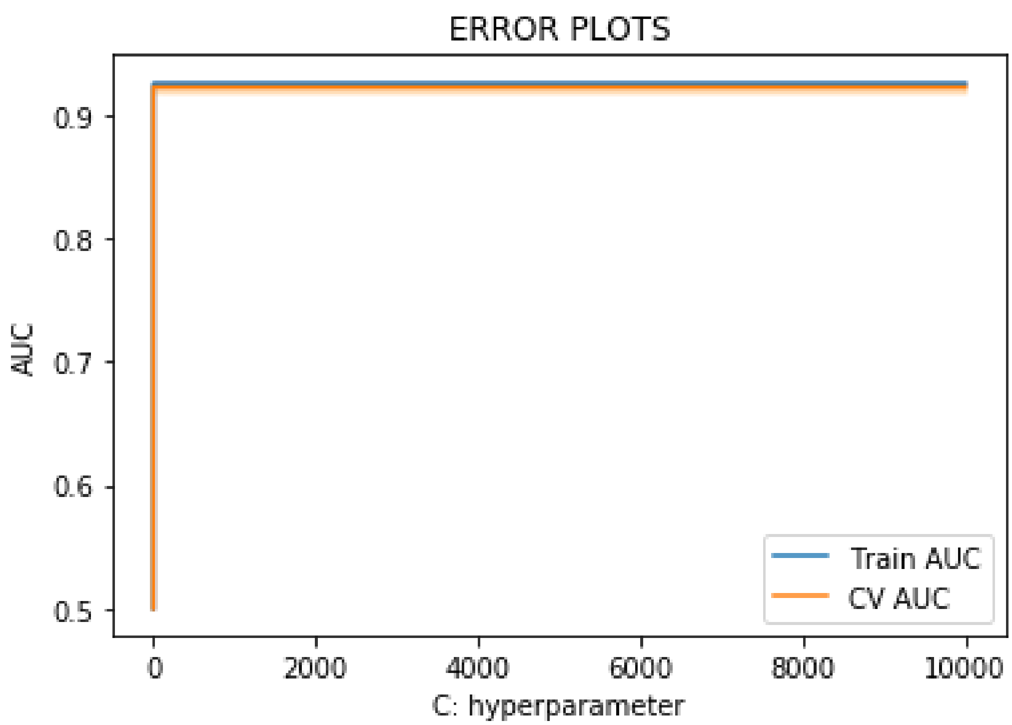
```
In [44]: def computeTfIdfW2v(data):

             list_of_sentence_train=[]
             for sentence in data:
                 list_of_sentence_train.append(sentence.split())

             model = TfidfVectorizer(ngram_range=(1,2), min_df=10)
             model.fit(data)
             # we are converting a dictionary with word as a key, and the idf as a value
             dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

              # this line of code trains your w2v model on the give list of sentances
             w2v_model=Word2Vec(list_of_sentence_train,min_count=20,size=50, workers=4, iter=1
             w2v_words = list(w2v_model.wv.vocab)

             # TF-IDF weighted Word2Vec
             tfidf_feat = model.get_feature_names() # tfidf words/col-names
             # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = t

             tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in th
             row=0;
             for sent in tqdm(list_of_sentence_train): # for each review/sentence
                 sent_vec = np.zeros(50) # as word vectors are of zero length
                 weight_sum =0; # num of words with a valid vector in the sentence/review
                 for word in sent: # for each word in a review/sentence
                     if word in w2v_words and word in tfidf_feat:
                         vec = w2v_model.wv[word]
                         #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                         # to reduce the computation we are
                         # dictionary[word] = idf value of word in whole courpus
                         # sent.count(word) = tf valeus of word in this review
                         tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                         sent_vec += (vec * tf_idf)
                         weight_sum += tf_idf
                 if weight_sum != 0:
                     sent_vec /= weight_sum
                 tfidf_sent_vectors.append(sent_vec)
                 row += 1
             return tfidf_sent_vectors
```

7.4.1 [5.4.1] Applying Logistic Regression with L1 regularization on TFIDF W2V, SET 4

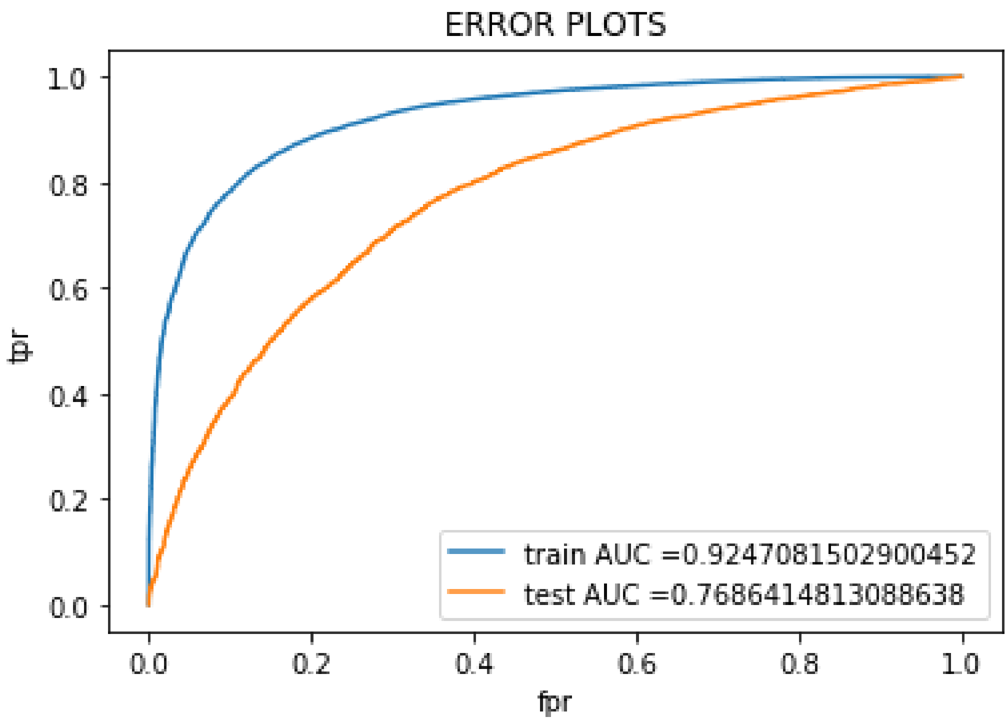
```
In [45]: X_train_w2vTfIdf = computeTfIdfW2v(X_train)
         X_test_w2vTfIdf = computeTfIdfW2v(X_test)

         clf = train_and_plot_auc('l1',C,X_train_w2vTfIdf, y_train)
```

```python
y_pred = clf.predict(X_test_w2vTfIdf)
print('Confusion Matrix : \n' + str(confusion_matrix(y_test,y_pred)))

predictAndPlot(X_train_w2vTfIdf, y_train, X_test_w2vTfIdf, y_test, clf)
print(clf.score(X_test_w2vTfIdf, y_test))
bestC = clf.best_params_
print(bestC)
w = clf.best_estimator_.coef_
print(np.count_nonzero(w))
```

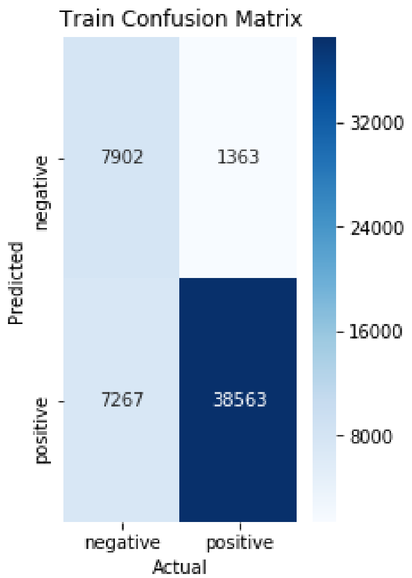
```
100%|| 55095/55095 [32:35<00:00, 28.17it/s]
100%|| 16456/16456 [01:28<00:00, 185.28it/s]
```

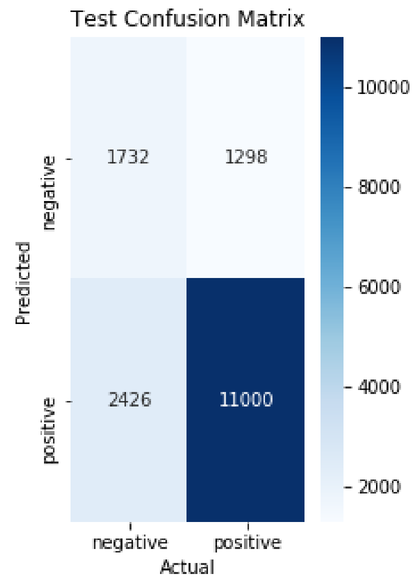



```
Confusion Matrix :
[[ 1732  1298]
 [ 2426 11000]]
```



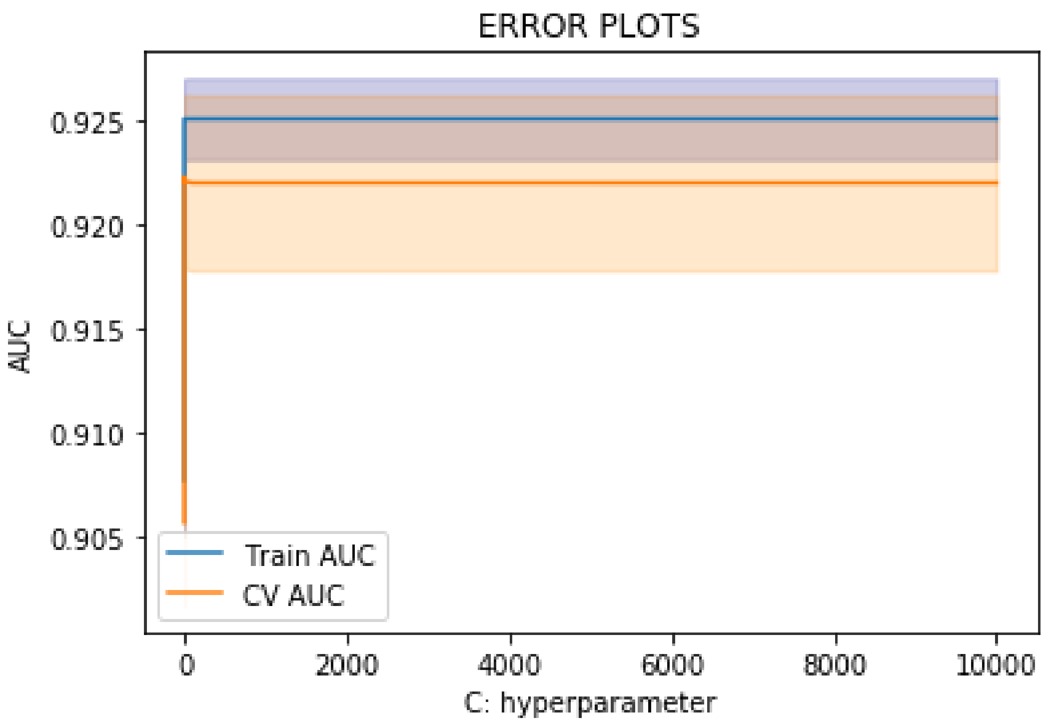

```
================================================================================
0.7686414813088638
{'C': 1}
49
```



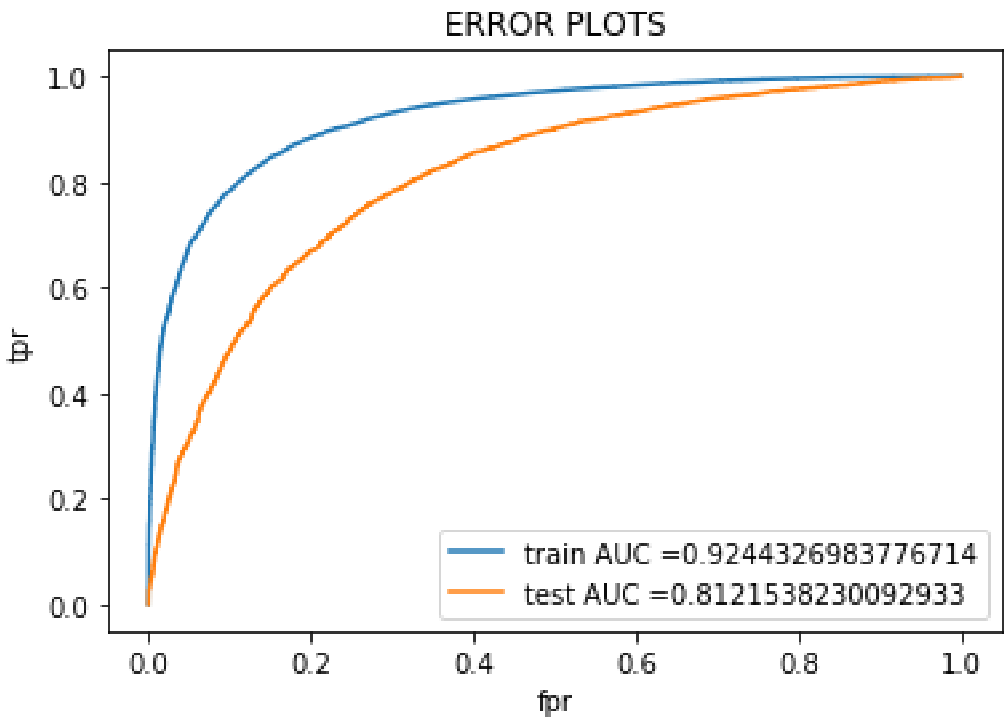


7.4.2 [5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

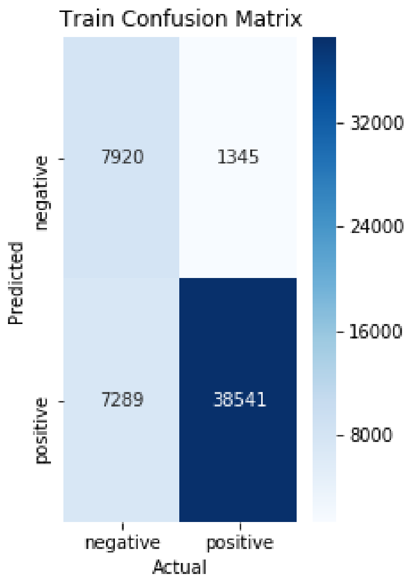
In [46]:
```
# Please write all the code with proper documentation
clf = train_and_plot_auc('l2',C,X_train_w2vTfIdf, y_train)
y_pred = clf.predict(X_test_w2vTfIdf)
print('Confusion Matrix : \n' + str(confusion_matrix(y_test,y_pred)))

predictAndPlot(X_train_w2vTfIdf, y_train, X_test_w2vTfIdf, y_test, clf)
print(clf.score(X_test_w2vTfIdf, y_test))
bestC = clf.best_params_
print(bestC)
w = clf.best_estimator_.coef_
print(np.count_nonzero(w))
```

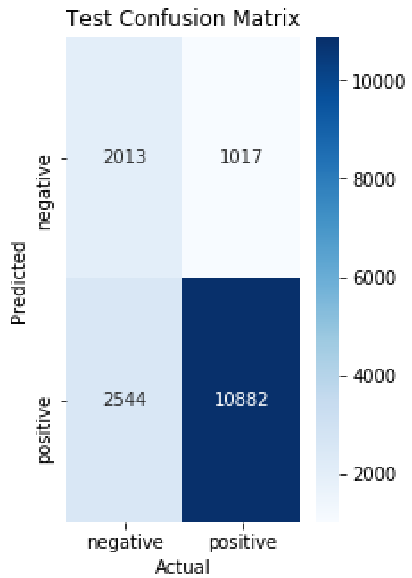



```
Confusion Matrix :
[[ 2013  1017]
 [ 2544 10882]]
```




```
================================================================================
0.8121538230092933
{'C': 0.01}
50
```





8 [6] Conclusions

```
In [50]: from prettytable import PrettyTable

         x = PrettyTable()
         x.field_names = ["Vectorizer", "Algorithm", "Reg", "HyperParameter", "Test AUC", "Trai
         x.add_row(["BOW", "LR", "l1", 1, 0.931, 0.991, "100k", 1])
         x.add_row(["BOW", "LR", "l2", 0.01, 0.935, 0.957, "100k", 1])
         x.add_row(["TFIDF", "LR", "l1", 1, 0.960, 0.978, "100k", 10])
         x.add_row(["TFIDF", "LR", "l2", 1, 0.964, 0.987, "100k", 10])
         x.add_row(["AVGW2V", "LR", "l1", 1, 0.853, 0.920, "100k", 20])
         x.add_row(["AVGW2V", "LR", "l2", 1, 0.851, 0.920,"100k", 20])
         x.add_row(["TFIDFW2V", "LR", "l1", 1, 0.833,0.899, "100k", 20])
         x.add_row(["TFIDFW2V", "LR", "l2", 1, 0.831, 0.899, "100k", 20])
         print()
         print()
         print("Tabular Results: with TimeSeries Split")
         print(x)
```

```
Tabular Results: with TimeSeries Split
+------------+-----------+-----+----------------+----------+-----------+----------+------------
| Vectorizer | Algorithm | Reg | HyperParameter | Test AUC | Train AUC | DataSize | min_count/n
+------------+-----------+-----+----------------+----------+-----------+----------+------------
|    BOW     |     LR    |  l1 |       1        |  0.931   |   0.991   |   100k   |           1
|    BOW     |     LR    |  l2 |      0.01      |  0.935   |   0.957   |   100k   |           1
|   TFIDF    |     LR    |  l1 |       1        |   0.96   |   0.978   |   100k   |          10
|   TFIDF    |     LR    |  l2 |       1        |  0.964   |   0.987   |   100k   |          10
|   AVGW2V   |     LR    |  l1 |       1        |  0.853   |    0.92   |   100k   |          20
|   AVGW2V   |     LR    |  l2 |       1        |  0.851   |    0.92   |   100k   |          20
|  TFIDFW2V  |     LR    |  l1 |       1        |  0.833   |   0.899   |   100k   |          20
|  TFIDFW2V  |     LR    |  l2 |       1        |  0.831   |   0.899   |   100k   |          20
+------------+-----------+-----+----------------+----------+-----------+----------+------------
```

```
In [51]: from prettytable import PrettyTable

         x = PrettyTable()
         x.field_names = ["Vectorizer", "Algorithm", "Reg", "HyperParameter", "Test AUC", "Trai
         x.add_row(["BOW", "LR", "l1", 1, 0.932, 0.991, "100k", 1])
         x.add_row(["BOW", "LR", "l2", 0.01, 0.937, 0.956, "100k", 1])
         x.add_row(["TFIDF", "LR", "l1", 1, 0.960, 0.977, "100k", 10])
         x.add_row(["TFIDF", "LR", "l2", 1, 0.964, 0.987, "100k", 10])
         x.add_row(["AVGW2V", "LR", "l1", 1, 0.867, 0.918, "100k", 20])
         x.add_row(["AVGW2V", "LR", "l2", 100, 0.866, 0.918, "100k", 20])
         x.add_row(["TFIDFW2V", "LR", "l1", 1, 0.793, 0.897, "100k", 20])
         x.add_row(["TFIDFW2V", "LR", "l2", 1, 0.793, 0.897, "100k", 20])
```

```python
print()
print()
print("Tabular Results: with Randomized Split (Shuffle=True)")
print(x)
```

Tabular Results: with Randomized Split (Shuffle=True)

Vectorizer	Algorithm	Reg	HyperParameter	Test AUC	Train AUC	DataSize	min_count/
BOW	LR	l1	1	0.932	0.991	100k	1
BOW	LR	l2	0.01	0.937	0.956	100k	1
TFIDF	LR	l1	1	0.96	0.977	100k	10
TFIDF	LR	l2	1	0.964	0.987	100k	10
AVGW2V	LR	l1	1	0.867	0.918	100k	20
AVGW2V	LR	l2	100	0.866	0.918	100k	20
TFIDFW2V	LR	l1	1	0.793	0.897	100k	20
TFIDFW2V	LR	l2	1	0.793	0.897	100k	20

In [52]:
```python
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Algorithm", "Reg", "HyperParameter", "Test AUC", "Tra
x.add_row(["BOW", "LR", "l1", 1, 0.954, 0.995, "100k", 1])
x.add_row(["BOW", "LR", "l2", 0.01, 0.957, 0.972, "100k", 1])
x.add_row(["TFIDF", "LR", "l1", 1, 0.973, 0.986, "100k", 10])
x.add_row(["TFIDF", "LR", "l2", 1, 0.976, 0.992, "100k", 10])
x.add_row(["AVGW2V", "LR", "l1", 1, 0.872, 0.946, "100k", 20])
x.add_row(["AVGW2V", "LR", "l2", 100, 0.871, 0.946, "100k", 20])
x.add_row(["TFIDFW2V", "LR", "l1", 1, 0.768, 0.924, "100k", 20])
x.add_row(["TFIDFW2V", "LR", "l2", 1, 0.812, 0.924, "100k", 20])
print()
print()
print()
print()
print()
print()
print("Tabular Results: TimeSeries Split and ReviewText + ReviewSummary as Features")
print(x)
```

Tabular Results: TimeSeries Split and ReviewText + ReviewSummary as Features

Vectorizer	Algorithm	Reg	HyperParameter	Test AUC	Train AUC	DataSize	min_count/r
BOW	LR	l1	1	0.954	0.995	100k	1
BOW	LR	l2	0.01	0.957	0.972	100k	1
TFIDF	LR	l1	1	0.973	0.986	100k	10
TFIDF	LR	l2	1	0.976	0.992	100k	10
AVGW2V	LR	l1	1	0.872	0.946	100k	20
AVGW2V	LR	l2	100	0.871	0.946	100k	20
TFIDFW2V	LR	l1	1	0.768	0.924	100k	20
TFIDFW2V	LR	l2	1	0.812	0.924	100k	20

9 Observations

1. With BOW and TFIDF featurization Logistic Regression performs slightly better than Naive Bayes Classifier.
2. The Best score observed is for TFIDF with L1 or L2 regularization.
3. The Top 10 +ve and -ve feature importances with BOW and TFIDF featurization look much more sensible with Logistic Regression than with NaiveBayes.
4. For Both AVG and TFIDF W2V featurization the CV performance seemed to closely follow the Train Performance.

   4.1 However the Train performance was found to be lower than in case of BOW and TFIDF.
   4.2 And the difference between Train and Test AUC in case of W2V models is also slightly larger then the corresponding difference in case of BOW and TFIDF.
5. Tried to experiment with the W2V featurization by changing values for min_count and iter. Tried values of 1, 5, 10, 20, 200 for min_count and values 1, 10, 15 for iter. And i have reported the values for the best combination found. The reported values are slightly better than then defaults min_count=5 and iter=1. But i was unable to get the performance closer to BOW/TFIDF.
6. Time series split and randomized split shows mixed results but overall the performance is approximately the same.
7. Improving the Feature Vector by adding Review Summary to Review Text clearly shows improved performance in both Test and Train, for all the featurizations.

Reasons for Poor Performance of W2V over Plain BOW/TFIDF

1. On the web i found one reference that says if Context is Domain specific then W2V may perform poorer than plain BOW/TFIDF. However for AFR Dataset i am not sure if context is the issue.
2. TODO : Still exploring other reasons.